

Re: Memory problem with XML::DOM::Parser???

Source: <http://coding.derkeiler.com/Archive/Perl/comp.lang.perl.misc/2004-06/0542.html>

From: Markus Mohr (markus.mohr_at_mazimoi.de)

Date: 06/07/04

Date: Mon, 07 Jun 2004 06:00:27 +0200

On Sun, 6 Jun 2004 03:52:10 +0000 (UTC), Ben Morrow
<usenet@morrow.me.uk> wrote:

```
>
>Quoth markus.mohr@mazimoi.de:
>>
>> Now, here is the code, and that's pretty all I have to master.
>>
>> Do you think there is anything to do about rewriting this piece of code
>> for XML::LibXML2?
>>
>> ----- Code sample -----
>> #!/usr/bin/perl -w
>
><standard moan>
>use strict;
>use warnings;
>
>>
#-----
>> # CFilter.pm
>> #
>> #
>> #
>> # Modul für die Filter-Funktionen des Client im Zusammenspiel mit
>> CGI.pm und #
>> # CXML.pm
>> #
>>
#-----
>
>Big box comments like this really don't help readability; and info about
>what the module is and does should be put in POD so it can be read later
>more easily.
>
>> use CXML;
>
>What is this module? It's not on CPAN, so I presume it's yours? By the
```

comp.lang.perl.misc: Re: Memory problem with XML::DOM::Parser???

```
>looks of things this will need rewriting as well.
>
>> # Pragmata
>> use diagnostics;
>> use strict;
>
>Oh right, you've got it down here... use strict and warnings should come
>first.
>
>> use open ':utf8';
>
>If you say
>
>use open ':encoding(utf8)';
>
>you will get better error handling and fallback facilities when the data
>isn't valid.
>
>> return 1;
>
>Don't do this... put it at the end.
>
>> sub import_anfrage ($$) {
>> my ( $self, $anfrage, $konfiguration ) = @_ ;
>> print "\nDie ANFRAGE wird importiert:\n";
>> print "-----\n";
>>
>> open( TEMP, ">./anf_temp.anf" );
>> print TEMP $anfrage;
>> close TEMP;
>
>You don't need to do this. XML::DOM and XML::LibXML can both parse XML
>from a string (though I admit that in the case of XML::DOM the
>documentation is less than clear...).
>
>> # Wir legen ein neues XML-Objekt an, das alte wird verworfen
>> my $xml = CXML->new();
>> $xml->construct_xml($konfiguration);
>> $xml = $konfiguration->get_value('xml');
>> my $xml_root = $xml->{'root'};
>
>Here is your first problem. CXML objects appear to contain XML::DOM
>objects; AFAIK there is no way to transfer a node from an XML::LibXML
>tree to an XML::DOM tree short of serialising it and re-parsing. This
>means you will have to modify CXML to use XML::LibXML (or whatever) as
>well.
>
>> # Die Anfrage wird in ein XML-Dokument geparkt
>> print "Debug: -> Die ANFRAGE wird gePARSt.\n";
>
>Debug messages like this are better sent to stderr with warn.
```

```
>
>> unlink("./anf_temp.anf");
>
>... or die translate_to_German("couldn't delete anf_temp.anf: $!");
>
>> # Die Anfrage ist Teil der neuen EPA
>> my $anfrage_root = $anfrage_doc->getElementsByTagName('ANFRAGE');
>> $anfrage_root = $anfrage_root->item(0);
>> $anfrage_root->setOwnerDocument( $xml->{'doc'} );
>> my $nodes = $xml_root->getElementsByTagName('anfragen');
>> my $node = $nodes->item(0);
>> $node->appendChild($anfrage_root);
>
>All of this stuff will be the same with XML::LibXML, once you have your
>CXML object using the same DOM library.
>
>In theory, as the DOM provides a specification of the methods etc., you
>should simply be able to switch 'XML::LibXML' for 'XML::DOM' throughout
>and it'll all be fine... it won't, of course (life's never that simple),
>but the changes required shouldn't be major.
>
>Ben
```

Okay, Ben, thank you very much. Here is the complete code for "CXML.pm" for your interest. Of course, it contains XML::DOM statements.

Can you have a look at this file as well?

```
-----
#!/usr/bin/perl -w

#-----#
# CXML.pm
#
#
# Modul für die XML-Funktionen des Clients
#
#-----#

package CXML;

#-----#
# Interne Versionierung
#
#-----#
use vars qw/$VERSION $TIMESTAMP/;
# $VERSION = 1.0;
# $TIMESTAMP = 20030321;
# $VERSION = 1.1;
```

comp.lang.perl.misc: Re: Memory problem with XML::DOM::Parser???

```
# $TIMESTAMP = 20030627;
# $VERSION = "1.5.4";
# $TIMESTAMP = 20040505;
# $VERSION = "1.5.5.build.1";
# $TIMESTAMP = 20040521;
$VERSION = "1.5.5.build.2";
$TIMESTAMP = 20040604;

#-----#
# Laden der internen Module (1)
#
#-----#
# XML::DOM ist ein CPAN Modul und existiert in dieser Form nicht auf
# ActiveState, sondern nur unter cpan.perl.org.
# XML::DOM muss daher auf dem Client-Rechner installiert sein!
use XML::DOM;

#-----#
# Laden der externen Module (0)
#
#-----#

# Pragmata
use diagnostics;
use strict;
use locale;

# use open ':utf8';

return 1;

#-----#
# Subroutine zum Anlegen einer neuen "Fallmappe"
#
#-----#
sub new {
    my $self = {};

    $self->{doc} = XML::DOM::Document->new();
    $self->{xml} = $self->{doc}->createXMLDecl( '1.0', 'UTF-8' );
    $self->{root} = undef;
    $self->{type} = undef;
    $self->{template} = undef;
    $self->{arzt} = undef;

    bless($self);
    return $self;
}

#-----#
# Subroutine, um die XML-Struktur aus dem XML-Rootfile und den
```

```

referenzierten #
# Dateien zu generieren
#
#-----#
sub construct_xml ($) {
    my ( $self, $konfiguration ) = @_ ;

    my $rootfile = $konfiguration->get_value('xmlrootfile');
    my $gui = $konfiguration->get_value('gui');
    my @xmlfiles = $rootfile;
    my %xmlroots;
    my %xmldocs;

    foreach my $current_file (@xmlfiles) {
        if ( -r $current_file ) {
            if ($gui) { $gui->set_status( 52, $current_file );
$gui->{main}->Busy( -recurse => 1 ); }
            else { print CText->get( $konfiguration, 52, $current_file
), "\n"; }
                open( XML, $current_file ) or die CText->get(
$konfiguration, 1001, $current_file );
                my @file = <XML>;
                my $line_tot = @file;
                close(XML);

                # Für jede XML-Datei einen Datenbaum erstellen
                my $xml_cur_doc = XML::DOM::Document->new();
                $xml_cur_doc->createXMLDecl( '1.0', 'UTF-8' );
                my $xml_cur_roo = undef;
                my @parent_list = ();

                # Die XML-Datei auswerten
                for ( my $line_cur = 0 ; $line_cur < $line_tot ;
$line_cur++ ) {
                    SWITCH: for ( $file[$line_cur] ) {

                        # Importierte XML-Schemata vormerken und später
einlesen
                            /include schemaLocation=("[\w\|.]+)"/ && do { my
$filename = substr( $current_file, 0, rindex( $current_file, "/" ) + 1
) . $1; push ( @xmlfiles, $filename ); last; };

                            # Ein </element>-Tag schliesst ein Wrapper-Element
                            /<\. *element>/ && do { my $x = shift
(@parent_list); last; };

                            # Referenz auf weitere Elemente/Datei überspringen
                            /element ref=("[\w+]")/ && do { last; };

                            # Normales Element – unter seinem Parent einordnen
und den Typ speichern

```

comp.lang.perl.misc: Re: Memory problem with XML::DOM::Parser???

```
/element name="\s*(\w+)\s*".*type="\s*?:(\w+)\s*" / &&
do {
    my $child = $xml_cur_doc->createElement($1);
    my ($parent) = @parent_list;
    $parent->appendChild($child);
    $self->{type}{$1} = $2;
    last;
};

# Komplexes oder Wrapper-Element
/element name="\s*(\w+)\s*" / && do {
    my $element = $1;

    # Falls in den nächsten Zeilen "complexType"
    und "Content" stehen ist es ein komplexes Element
    if ( $file[ $line_cur + 1 ] =~ /complexType/
    && $file[ $line_cur + 2 ] =~ /Content/ ) {
        my $child =
$xml_cur_doc->createElement($element);
        my ($parent) = @parent_list;
        $parent->appendChild($child);
        my @enum_values = ();

        until ( $file[ $line_cur - 1 ] =~
/<\/.*element>/ ) {
            if ( $file[$line_cur] =~ /enumeration
value="\s*(.*?)\s*" ) { push ( @enum_values, $1 ); }
            $line_cur++;
        }
        $self->{type}{$element} = "enum";
        $self->{enum}{$element} = [ @enum_values ];
        last;

        # Ansonsten ist es ein Wrapper-Element das
als Parent fungiert
    }
    else {
        my $parent =
$xml_cur_doc->createElement($element);
        if ( defined $xml_cur_roo ) { my
($preparent) = @parent_list; $preparent->appendChild($parent); }
        else { $xml_cur_roo = $parent; }
        unshift ( @parent_list, $parent );
        last;
    }
}
}
```

Das erzeugte XML-Dokument für diese Datei in einem Hash
ablegen – Index ist der Dateiname

comp.lang.perl.misc: Re: Memory problem with XML::DOM::Parser???

```
$self->{template}{doc}{$current_file} = $xml_cur_doc;
$self->{template}{root}{$current_file} = $xml_cur_roo;

}
else {
    die CText->get( $konfiguration, 1001, $current_file );
}
}

$self->{template}{root}{$rootfile}->setOwnerDocument( $self->{doc}
);
$self->{root} = $self->{template}{root}{$rootfile};

# In die Konfiguration die Referenz auf das XML-Objekt ablegen
$konfiguration->set_value( 'xml', $self );

# Einen Patienten anlegen
CXML->insert( $konfiguration, 'pat' );

if ($gui) { $gui->set_status(53); $gui->{main}->Unbusy; }
return;
}

#-----#
# Subroutine, um XML-File aus einer Datei einzulesen
#
#-----#
sub read($$) {
    my ( $self, $file, $konfiguration ) = @_ ;

    if ( $konfiguration->get_value('gui') ) { return if
$konfiguration->get_value('gui')->create_confirm( CText->get(
$konfiguration, 950 ), $konfiguration ); }
    if ( $konfiguration->get_value('gui') ) {
$konfiguration->get_value('gui')->set_status(54); }

    my $parser = new XML::DOM::Parser( KeepCDATA => 1, ErrorContext =>
2 );
    my $doc = $parser->parsefile($file);
    unless ($doc) {
        warn CText->get( $konfiguration, 1002 );

        # Logfileeintrag
        if ( defined $konfiguration->get_value('log') ) { CTools->log(
$konfiguration, 904 ); }
    }
    my $xml = $konfiguration->get_value('xml');
    $xml->{'doc'} = $doc;
    $xml->{'root'} = $doc;
    close(XML);
}
```

comp.lang.perl.misc: Re: Memory problem with XML::DOM::Parser???

```
# Nach dem Import werden in der internen Datendarstellung die
Umlaute als
# Umlaute und nicht codiert gefuehrt
for my $schild ( $xml->{'root'}->getElementsByTagName('*') ) {
    if ( $schild->toString =~ /<![CDATA\[([.*?&#\d{3};*?)]\]>/ )
    {
        my $schilddata = $1;
        my @list = $schild->getElementsByTagName('*');
        if ( $#list eq -1 ) {
            $schilddata = CXML->code($schilddata);
            my $value_node =
$xml->{'doc'}->createCDATASection($schilddata);
            my $fc = $schild->getFirstChild;
            $schild->replaceChild( $value_node, $fc );
        }
    }
}

# Logfileeintrag
if ( defined $konfiguration->get_value('log') ) { CTools->log(
$konfiguration, 905 ); }
return;
}

# -----#
# Subroutine, um die XML-Struktur in eine Datei zu schreiben
#
# -----#
sub write($$) {
    my ( $self, $konfiguration ) = @_ ;

    # Dateinamen ermitteln, dazu ID ds angemeldeten Arztes, Vorname,
Nachname
    # und Geburtsdatum ermitteln
    my $arzt_id = $konfiguration->get_value('uid');
    my $pat_data = CXML->extract_flattened( 'VCARDMOD', 'PATIENT', 0,
1, $konfiguration );
    my $soz_data = CXML->extract_flattened( 'soziomedizinischedaten',
", 0, 0, $konfiguration );
    $pat_data =~
/<id>(d+)</id>.*?<vorname><![CDATA\[([.*?)]\]></vorname>.*?<nachname><![CDATA\[([.*?)]\]></nachname>
return 0 unless $1 && $2 && $3;
    my $file = join ( "-", ( $arzt_id, $1, $2, $3 ) );
    $soz_data =~ /<geburtszeitpunkt>(d+)-(d+)-(d+)/s;
    return 0 unless $1 && $2 && $3;
    $file .= "-$3-$2-$1.epa";

    # Datei zum schreiben öffnen
    open( XML, ">$file" ) or die CText->get( $konfiguration, 1001,
$file );
```

```

# EPA auf Festplatte bringen
print XML xmlcode(
$konfiguration->get_value('xml')->{'root'}->toString );
    close(XML);
    if ( defined $konfiguration->get_value('log') ) { CTools->log(
$konfiguration, 906 ); }
    return 1;
}

#-----#
# Subroutine, um die bezeichnete Patientenakte von der Festplatte zu
entfernen #
#-----#
sub delete($$) {
    my ( $self, $filename, $konfiguration ) = @_;
    if ( $konfiguration->get_value('gui') ) { return if
$konfiguration->get_value('gui')->create_confirm( CText->get(
$konfiguration, 951 ), $konfiguration ); }

    unlink $filename;

    if ( defined $konfiguration->get_value('log') ) { CTools->log(
$konfiguration, 907, $filename ); }
    if ( $konfiguration->get_value('gui') ) {
$konfiguration->get_value('gui')->set_status(55); }
    return;
}

#-----#
# Subroutine, um einen bestehenden Ast des XML-Schemas in einen
anderen zu #
# kopieren bzw. zu bewegen
#
#-----#
sub copy ($$$$$$) {
    my ( $self, $from_major, $from, $from_id, $to, $to_id,
$to_element, $remove_source, $konfiguration ) = @_;

    my $xml = $konfiguration->get_value('xml');
    my $xmlroot = $xml->{'root'};

    # Source-Bereich finden
    for my $source ( $xmlroot->getElementsByTagName($from_major) ) {
        for my $source ( $source->getElementsByTagName($from) ) {

            # Anschliessend die ID des Bereiches suchen
            for my $id_node ( $source->getElementsByTagName('id') ) {

                # Und feststellen ob es die gewünschte ID ist
                if ( $id_node->getFirstChild->toString =~
/^$from_id$/i ) {

```

comp.lang.perl.misc: Re: Memory problem with XML::DOM::Parser???

```
# Falls dem so sein sollte, den Bereich (rekursiv)
kopieren...
my $nodecopy = $source->cloneNode(1);

# Eventuell die alte Node entfernen
if ($remove_source) { my $source_parent =
$source->getParentNode(); $source_parent->removeChild($source); }

# ...anschliessen den Zielabschnitt suchen
for my $destination (
$xmlroot->getElementsByTagName($to) ) {

    # Anschliessend die ID des Ziels suchen
    for my $id_node (
$destination->getElementsByTagName('id') ) {

        # Und feststellen ob es die gewünschte ID
ist
        if ( $id_node->getFirstChild->toString =~
/^$to_id$/i ) {
            for my $destination (
$destination->getElementsByTagName($to_element) ) {

                # Falls dem so sein sollte, den
kopierten Bereich anfügen
                $destination->appendChild($nodecopy);
                return;
            }
        }
    }
}
}
}
}
}
return;
}

# -----#
# Subroutine, um ein Wert-Child zu entfernen
#
# -----#
sub remove ($$$$) {
    my ( $self, $abschnitt, $keyword, $element, $nr, $konfiguration )
= @_ ;

    # Das XML-Objekt holen
    my $xml = $konfiguration->get_value('xml');
    my $xmlroot = $xml->{'root'};
    my ( $abs, @values );
```

comp.lang.perl.misc: Re: Memory problem with XML::DOM::Parser???

```
# Zuerst den Abschnitt suchen (z.B. 'anfragen')
if ($abschnitt) { ( $abs, @values ) =
$xml->{'root'}->getElementsByTagName($abschnitt); }
else { $abs = $xml->{'root'}; }

# Anschliessend den Unterabschnitt suchen (z.B. eine 'ANFRAGE')
my $nodes = $abs->getElementsByTagName($keyword);

# Falls es mehrere Unterabschnitte gibt den gewünschten auswählen
my $teil = $nodes->item($nr);
if ( defined $teil ) {

    # Rekursiv im Unterabschnitt nach dem Tag dessen Wert gelöscht
    werden soll suchen (z.B. fachrichtung)
    for my $elem ( $teil->getElementsByTagName( $element, 1 ) ) {

        # Falls dieser Tag ein Wert-Kind besitzt dieses löschen
        my $value_child = $elem->getFirstChild;
        $elem->removeChild($value_child) if defined $value_child;
    }
}

return;
}

#-----#
# Subroutine, um die uebergebene XML-Struktur in das Gesamtschema
# einzupflegen #
#-----#
sub insert($$$) {
    my ( $self, $konfiguration, $insert_this, $parent ) = @_ ;

    my $xml = $konfiguration->get_value('xml');
    my $xmlroot = $xml->{'root'};
    my $xmlrootfile = $konfiguration->get_value('xmlrootfile');

    SWITCH: for ($insert_this) {
        /pat/ && do { $insert_this = "PATIENT"; $parent =
"patient"; last; };
        /arz/ && do { $insert_this = "ARZT"; $parent =
"arztliste"; last; };
        /par/ && do { $insert_this = "INSTITUTION"; $parent =
"paramedizinischeliste"; last; };
        /ana/ && do { $insert_this = "ANAMNESE"; $parent =
"anamnesen"; last; };
        /unt/ && do { $insert_this = "UNTERSUCHUNG"; $parent =
"untersuchungen"; last; };
        /dia/ && do { $insert_this = "DIAGNOSE"; $parent =
"diagnosen"; last; };
        /mas/ && do { $insert_this = "MASSNAHME"; $parent =
"massnahmen"; last; };
    }
}
```

comp.lang.perl.misc: Re: Memory problem with XML::DOM::Parser???

```
/anf/ && do { $insert_this = "ANFRAGE"; $parent =
"anfragen"; last; };
}

for my $parent_element ( $xmlroot->getElementsByTagName($parent) )
{

    # Dateinamen für das einzusetzende Datenblatt bestimmen, da
der
    # Dateiname als Index dient
    my $insert_file = $xmlrootfile;
    $insert_file =~ s/(.*\v)\w+(\.xsd)/$1$insert_this$2/i;

    # Neue ID für das Element generieren
    my $newid = 1;
    my %oldid;

    # Vorhandenen ID-Nodes suchen
    my @id_nodes = $parent_element->getElementsByTagName( 'id', 1
);
    my $id_anz = $#id_nodes;

    # Wenn die Anzahl -1 ist, gibt es keine IDs
    unless ( $id_anz < 0 ) {

        # Vorhandene IDs auslesen
        foreach my $id_node (@id_nodes) {
            if ( $id_node->hasChildNodes ) { my $id =
$id_node->getFirstChild->toString; $oldid{$id} = 1; }
            }
            $newid++ while $oldid{$newid};
        }

        # Neues Element erzeugen
        my $neuelement =
$xml->{template}{root}{$insert_file}->cloneNode(1);
        $neuelement->setOwnerDocument( $xml->{doc} );

        # Und die ID des Elementes setzen
        for my $id_node ( $neuelement->getElementsByTagName( 'id', 1 )
) { my $id_value = $xml->{doc}->createTextNode($newid);
$id_node->appendChild($id_value); }

        # Falls die Arzt-ID als Erzeuger-ID gesetzt werden kann, dies
tun
        for my $id_node ( $neuelement->getElementsByTagName( 'arztid',
1 ) ) { my $id_value = $xml->{doc}->createTextNode(
$konfiguration->get_value('uid') ); $id_node->appendChild($id_value);
}
}
```

comp.lang.perl.misc: Re: Memory problem with XML::DOM::Parser???

```
# Neues Element in den XML-Baum einpflegen
$parent_element->appendChild($neuelement);
}

return;
}

#-----#
# Subroutine, um den Wert eines Elementes in einem XML-Abschnitt zu
# aendern #
#-----#
sub update($$$$$) {
    my ( $self, $element, $wert, $keyword, $abschnitt, $nr,
    $konfiguration ) = @_;

    print "ICH SOLL UPDATEN: ELEMENT $element AUF WERT $wert KEYWORD
    $keyword ABSCHNITT $abschnitt NR $nr...\n";

    # Das XML-Objekt holen
    my $xml = $konfiguration->get_value('xml');

    # Variablen der Subroutine deklarieren
    my $count = 0;

    # Den jeweiligen Abschnitt suchen
    if ($abschnitt) { my @abschnitte =
    $xml->{'root'}->getElementsByTagName($abschnitt); $abschnitt =
    $abschnitte[0]; }
    else { $abschnitt = $xml->{'root'}; }

    print "SUCHE TEILABSCHNITT $nr, bin bei $count...\n";

    # Suchen wir nach dem Element, das mit dem Keyword bezeichnet wird
    for my $teil ( $abschnitt->getElementsByTagName($keyword) ) {

        # Prüfen ob wir auch das richtige Keyword gefunden haben (z.B.
        die VCARDMOD des 3. Arztes)
        if ( $count eq $nr ) {

            print "GEFUNDEN!\n";

            # ...und dann den Tag sofern er existiert
            for my $zieltag ( $teil->getElementsByTagName($element) )
            {

                print "ZIELTAG GEFUNDEN\n";

                # Das Werte-Element des Zieltags erzeugen
                my $new_value_element = CXML->create_value_element(
                $wert, $zieltag, $xml, $konfiguration ) if $wert;
            }
        }
    }
}
```

comp.lang.perl.misc: Re: Memory problem with XML::DOM::Parser???

```
# Hat das Zielement bereits ein Value-Kind?
if ( defined $zieltag->getFirstChild ) {

    print "HAT CHILD\n";

    # Wenn es bereits ein Value-Kind gibt das alte
    # ersetzen bzw. löschen falls Wert " ist
    my $old_value_element = $zieltag->getFirstChild;
    if ( defined $wert && defined $new_value_element
    && $wert ) {

        print "REPLACED\n";
        $zieltag->replaceChild( $new_value_element,
        $old_value_element );
    }
    else {
        $zieltag->removeChild($old_value_element);

        print "REMOVED\n";
    }
}
else {
    $zieltag->appendChild($new_value_element) if $wert
    && defined $new_value_element;

    print "APPEND\n";
}
}
return 1;
}

# Wir gehen weiter in der Liste und suchen das naechste
# Vorkommen (z. B.
# die VCARDMOD des naechsten Arztes)
$count++;
}

print "FERTIG!\n";

return 0;
}

#-----#
# Subroutine, um aus dem Gesamtschema den durch das Keyword
# beschriebenen Teil #
# auszulesen
#
#-----#
sub extract($$$$) {
    my ( $self, $keyword, $abschnitt, $nr, $konfiguration ) = @_;
```

```

# Das XML-Objekt holen
my $xml = $konfiguration->get_value('xml');
my $abs;

# Variablen der Subroutine deklarieren
my $count = 0;
my @values;

if ($abschnitt) { ( $abs, @values ) =
$xml->{'root'}->getElementsByTagName($abschnitt); }
else { $abs = $xml->{'root'}; }

my $nodes = $abs->getElementsByTagName($keyword);

my $total = $nodes->getLength;

my $teil = $nodes->item($nr);

my %valueshash;
if ( defined $teil ) {

    # Alle Nodes aus dem Abschnitt holen
    sub getallchildnodes {
        my ( $node, $parentname, $valuesref, $hashref ) = @_ ;

        # Child-Nodes jeder Node durchlaufen
        foreach my $child ( $node->getChildNodes ) {

            # Falls auch diese Node Kinder hat, rekursiv
            durchlaufen
            getallchildnodes( $child, $child->getNodeName,
            $valuesref, $hashref ) if $child->hasChildNodes;

            # Keine Kinder? Dann ist es eine Wert-Node - Wert
            ermitteln bzw. " setzen falls nicht initialisiert
            my $value = defined $child->getNodeValue ?
            $child->getNodeValue : "";

            # Den Wert speichern unter dem Namen der Eltern-Node
            unless ( defined $hashref->{ $child->getNodeName } &&
            $hashref->{ $child->getNodeName } ne " ) { $hashref->{
            $child->getNodeName } = $value; }
            unless ( defined $hashref->{ $parentname } &&
            $hashref->{ $parentname } ne " ) { $hashref->{ $parentname } = $value; }
            }
        }
    }

    # Rekursive Funktion anstossen
    getallchildnodes( $teil, $teil->getNodeName, \@values,
    \%valueshash );
}

```

comp.lang.perl.misc: Re: Memory problem with XML::DOM::Parser???

```
# Hash neu strukturieren
foreach my $tagname ( keys %valueshash ) { push ( @values, {
$tagname => $valueshash{$tagname} } ) unless $tagname =~
/cdata-section/; }

# Array mit der Gesamtzahl und den Werten zurückliefern
return ( $total, @values );
}

#-----#
# Subroutine um aus dem Gesamtschema einen Teil als Scalar auszugeben
#
#-----#
sub extract_flattened ($$$$) {
    my ( $self, $keyword, $abschnitt, $nr, $id, $konfiguration ) = @_;

    # Das XML-Objekt holen
    my $xml = $konfiguration->get_value('xml');
    my ( $abs, @temp );

    # Variablen der Subroutine deklarieren
    my $count = 0;

    if ( $abschnitt ) { ( $abs, @temp ) =
$xml->{'root'}->getElementsByTagName($abschnitt); }
    else { $abs = $xml->{'root'}; }

    my $nodes = $abs->getElementsByTagName($keyword);

    # Suchen wir nach einer Nr oder einer ID?
    if ( $id == 0 ) {
        my $teil = $nodes->item($nr);
        if ( defined $teil ) { return $teil->toString; }
    }
    else {

        foreach my $teil ( @{$nodes} ) { return $teil->toString if
$teil->toString =~ /<id>$id</id>/; }
    }

    return;
}

#-----#
# Subroutine, um die gesamte EPA als String zu dumpen
#
#-----#
sub extract_all_flattened ($) {
    my ( $self, $konfiguration ) = @_;
    return $konfiguration->get_value('xml')->{'root'}->toString;
}
}
```

```

#-----#
# Subroutine, um die Daten für den Menuebotton einer Anfrage zu
# generieren #
#-----#
sub get_payload_data ($$$$$) {
    my ( $self, $nr, $konfiguration, $existref, $picksref, $beref ) =
    @_;

    # Das XML-Objekt holen
    my $xml = $konfiguration->get_value('xml');
    my $xmlroot = $xml->{'root'};

    # Die vorhandenen Anamnesen, Untersuchungen, Diagnosen und
    # Massnahmen suchen
    foreach my $area (qw/ANAMNESE UNTERSUCHUNG DIAGNOSE MASSNAHME/) {
        foreach my $entity ( $xmlroot->getElementsByTagName($area) ) {

            # Nach der ID suchen
            my $entity_text = $entity->toString;
            $entity_text =~ /<id>(\d+)</id>/;
            my $id = $1;
            push @ { $existref->{"\L$area\E"} }, $id;
            $picksref->{"\L$area\E"}->{$id} = 0;
            SWITCH: for ($area) {
                /ANAMNESE/ && do {
                    $entity_text =~
/<text>.*?CDATA\[([.*?])\].*?</text>.*?<anamnesezeitpunkt>(\d+)\D(\d+)\D(\d+)\s/;
                    my ( $b, $j, $m, $t ) = ( $1, $2, $3, $4 );
                    $b = "Anamnese" unless $b;
                    $j = "???" unless $j;
                    $m = "???" unless $m;
                    $t = "???" unless $t;
                    $beref->{'anamnese'}->{$id} = substr( code($b), 0,
40 ) . " ($t.$m.$j)";
                    last;
                };
                /UNTERSUCHUNG/ && do {
                    $entity_text =~
/<untersuchungsbezeichnung>([.*?])</untersuchungsbezeichnung>.*?<untersuchungszeitpunkt>(\d+)\D(\d+)\D(\d+)\s/;
                    my ( $b, $j, $m, $t ) = ( $1, $2, $3, $4 );
                    $b = "Untersuchung" unless $b;
                    $j = "???" unless $j;
                    $m = "???" unless $m;
                    $t = "???" unless $t;
                    $beref->{'untersuchung'}->{$id} = substr(
code($b), 0, 40 ) . " ($t.$m.$j)";
                    last;
                };
                /DIAGNOSE/ && do {
                    $entity_text =~
/<diagnosetyp>.*?CDATA\[([.*?])\].*?</diagnosetyp>.*?<diagnosezeitpunkt>(\d+)\D(\d+)\D(\d+)\s/;

```

```

my ( $b, $j, $m, $t ) = ( $1, $2, $3, $4 );
$b = "Diagnose" unless $b;
$j = "???" unless $j;
$m = "???" unless $m;
$t = "???" unless $t;
$beref->{'diagnose'}->{$id} = substr( code($b), 0,
40 ) . " ($t.$m.$j)";
    last;
};
/MASSNAHME/ && do {
    $entity_text =~
/<bezeichnung>.*?CDATA\[([.*?])\].*?</bezeichnung>.*?<zeitpunktbeginn>(\d+)\D(\d+)\D(\d+)\s/;
    my ( $b, $j, $m, $t ) = ( $1, $2, $3, $4 );
    $b = "Massnahme" unless $b;
    $j = "???" unless $j;
    $m = "???" unless $m;
    $t = "???" unless $t;
    $beref->{'massnahme'}->{$id} = substr( code($b),
0, 40 ) . " ($t.$m.$j)";
        last;
    };
}
}
}

```

```

# Feststellen, welche Daten als zu senden gespeichert sind
my $anfrageliste = $xmlroot->getElementsByTagName('ANFRAGE');
my $anfrage = $anfrageliste->item($nr);
my $datentransmit =
$anfrage->getElementsByTagName('datentransmit')->item(0) if defined
$anfrage;
my $idstring = $datentransmit->toString if defined
$datentransmit;

```

```

while ( defined $idstring && $idstring =~ /<(\w+)id>(\d+)</ / ) {
    my $type = $1;
    my $id = $2;
    $picksref->{$type}->{$id} = 1;
    $idstring =~ s/<($type)id>$id</($type)id>//;
}

```

```

return;
}

```

```

#-----#
# Subroutine, um die Daten für den Menuebutton einer Anfrage
festzulegen #
#-----#

```

```

sub set_payload_data ($$$$) {
    my ( $self, $nr, $konfiguration, $typ, $id, $status ) = @_ ;

```

```

# Das XML-Objekt holen
my $xml = $konfiguration->get_value('xml');
my $xmlroot = $xml->{'root'};

# Feststellen, welche Daten als zu senden gespeichert sind
my $anfrageliste = $xmlroot->getElementsByTagName('ANFRAGE');
my $anfrage = $anfrageliste->item($nr);
my $datentransmit =
$anfrage->getElementsByTagName('datentransmit')->item(0) if defined
$anfrage;
my $tagname = $typ . "id";

if ($status) {

    # ID zur Payload hinzufügen
    my $value_element = $xml->{'doc'}->createEl
$type eq 'base64Binary';

    # Die Encodings in Umlaute wandeln
    $import_value = code($import_value) unless $type eq
'base64Binary';

SWITCH: for ($type) {
    /string/ && do {
        if ( $import_value =~ /[w]/ ) { $value_node =
$xml->{'doc'}->createCDATASection($import_value) }
        last;
    };
    /dateTime/ && do {
        if ( $import_value =~ /[d|+]/ ) { $value_node =
$xml->{'doc'}->createTextNode( date_iso($import_value) ) }
        last;
    };
    /dateTimefix/ && do {
        if ( $import_value =~ /[d|+]/ ) { $value_node =
$xml->{'doc'}->createTextNode( date_iso($import_value) ) }
        last;
    };
    /long/ && do {
        if ( $import_value =~ ^\d*/ && $import_value < 2147483647
) { $value_node = $xml->{'doc'}->createTextNode($import_value) }
        last;
    };
    /int/ && do {
        if ( $import_value =~ ^\d*/ && $import_value < 32767 ) {
$value_node = $xml->{'doc'}->createTextNode($import_value) }
        last;
    };
    /enum/ && do {

```

comp.lang.perl.misc: Re: Memory problem with XML::DOM::Parser???

```
    foreach my $value_allowed ( @{ $xml->{enum}}{
$zieltag->getTagName } ) {
    if ( $import_value eq $value_allowed ) { $value_node =
$xml->{'doc'}->createTextNode($import_value); }
    elsif ( convert_value( $import_value, $value_allowed )
) { $value_node = $xml->{'doc'}->createTextNode($value_allowed); }
    }
    last;
};
/base64Binary/ && do { $value_node =
$xml->{doc}->createCDATASection($import_value); last; };
die CText->get( $konfiguration, 1003, $_ );
}
```

```
    return $value_node;
```

```
}
```

```
#-----#
```

```
# Subroutine, um einen Wert auf einen Enumerationswert hinzubiegen,
```

```
falls #
```

```
# maeglich - TRUE zurueckgeben, falls das geht, ansonsten FALSE
```

```
#
```

```
#-----#
```

```
sub convert_value ($$) {
```

```
    my ( $iv, $av ) = @_;
```

```
    SWITCH: for ( $av ) {
```